Le programme "hello world"

Un programme “hello world” est un programme qui affiche simplement le message “hello, world!” (d'où son nom). Ce type de programme est souvent utilisé comme premier exemple, pour montrer la syntaxe de base d'un langage de programmation. À partir de cet exemple, vous allez voir comment afficher un message en C++.

Le programme “hello world” en C++ est assez proche du programme minimal présenté dans le chapitre précédent : Tester ce code.

main.cpp

```
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
}
```

Par rapport au code minimaliste, vous pouvez voir que l'on a ajouté deux lignes : la ligne d'inclusion (la première ligne du code, commençant par `#include`) et la ligne pour afficher le message (la cinquième ligne, commençant par `std::cout`).

L'affichage proprement dit est réalisé en utilisant un objet particulier, nommé `cout`, fournit par la bibliothèque standard.

La bibliothèque standard

Lorsque l'on parle du C++, il faut en fait distinguer deux choses : le langage C++ et la bibliothèque standard. Le langage C++ proprement-dit propose uniquement les fonctionnalités fondamentales indispensables pour écrire un programme. L'utilisation du langage ne nécessite aucune inclusion supplémentaire, le compilateur se charge du travail.

Un grand nombre de fonctionnalités du C++ n'est pas pris en charge directement par le langage, mais par la bibliothèque standard. Celle-ci est en fait constituée d'un ensemble de codes écrit en C++, permettant de proposer de nombreuses les fonctionnalités avancées. Il est indispensable d'apprendre à utiliser la bibliothèque standard lorsque l'on apprend le C++, l'un ne va pas sans l'autre.

Pour utiliser les fonctionnalités de la bibliothèques standards, il faut dans un premier temps inclure le fichier contenant le code C++ correspondant à la fonctionnalité que vous souhaitez utiliser. L'inclusion d'un fichier se fait en utilisant la directive de pré-processeur `#include`, suivi du nom du fichier à inclure. Il existe deux syntaxes possibles :

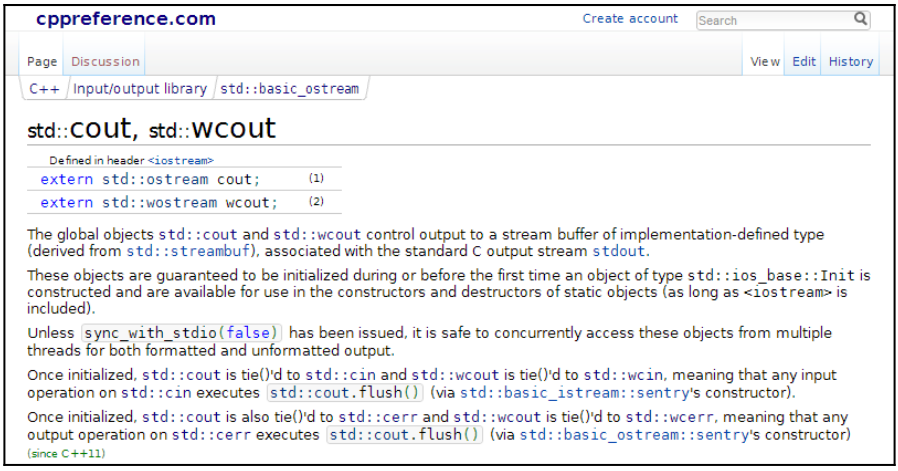
```
#include <nom_fichier>
#include "nom_fichier"
```

La première syntaxe permet d'inclure les fichiers correspondant à la bibliothèque standard, c'est ce qui nous intéresse ici. La seconde version vous permettra d'inclure vos propres fichiers, vous verrez dans la suite de ce cours comment créer des fichiers.

Les directives de pré-processeur

Une directive de pré-processeur permet de paramétrer le comportement du pré-processeur lors de la compilation. Il existe différentes directives, vous en verrez plusieurs dans ce cours. Une directive s'écrit toujours avec un dièse suivi de la directive et d'éventuels paramètres optionnels.

Pour afficher un message, il faut donc utiliser un objet particulier de la bibliothèque standard, nommé `cout`. Si vous regardez dans la documentation de cet objet, vous voyez au début de la page qu'il est écrit : “Defined in header <iostream>”.


cppreference.com

Create account Search

Page Discussion View Edit History

C++ / Input/output library / std::basic_ostream

std::cout, std::wcout

Defined in header <iostream>

extern std::ostream cout; (1)

extern std::wostream wcout; (2)

The global objects std::cout and std::wcout control output to a stream buffer of implementation-defined type (derived from std::streambuf), associated with the standard C output stream stdout.

These objects are guaranteed to be initialized during or before the first time an object of type std::ios_base::Init is constructed and are available for use in the constructors and destructors of static objects (as long as <iostream> is included).

Unless sync_with_stdio(false) has been issued, it is safe to concurrently access these objects from multiple threads for both formatted and unformatted output.

Once initialized, std::cout is tie()'d to std::cin and std::wcout is tie()'d to std::wcin, meaning that any input operation on std::cin executes std::cout.flush() (via std::basic_istream::sentry's constructor).

Once initialized, std::cout is also tie()'d to std::cerr and std::wcout is tie()'d to std::wcerr, meaning that any output operation on std::cerr executes std::cout.flush() (via std::basic_ostream::sentry's constructor)

(since C++11)

Cela vous indique quelle fichier il faut inclure pour utiliser `cout` : le fichier `iostream` :

```
#include <iostream>
```

“iostream” signifie “input output stream”, ce qui signifie “flux d'entrée et sortie”. “Entrée” et “sortie” doivent être compris du point de vue du programme : “entrée” d'information depuis l'extérieur vers l'intérieur du programme (par exemple saisie d'un texte par l'utilisateur ou la lecture d'un fichier) et “sortie” d'information depuis le programme vers l'extérieur (comme par exemple afficher un message à l'écran ou enregistrer une fichier). Vous verrez juste en dessous pourquoi on parle de “flux”.

Les flux standards

Si vous avez regardé un peu la documentation de `cout`, vous avez peut-être remarqué qu'il existe d'autres flux de sortie :

- `cout` et `wcout` pour les messages standard ;
- `cerr` et `wcerr` pour les messages d'erreur ;
- `clog` et `wclog` également pour les messages d'erreur.

Le `w` signifie que le flux prend en charge les caractères accentués...

Lorsque vous utiliserez une fonctionnalité de la bibliothèque standard que vous ne connaissez pas, vous pourrez de la même manière allez rechercher dans la documentation quel est le fichier à inclure.

L'espace de nom std

En C++, chaque chose doit avoir un nom unique, pour permettre au compilateur de les identifier correctement. Donner un nom n'est pas très compliqué, vous verrez par la suite les quelques règles à respecter. Lorsque l'on a un petit programme de quelques centaines ou milliers de ligne, cela pose pas trop de problème pour trouver des noms uniques. Mais dans le cas d'un programme de plus grande taille ou utilisant différentes bibliothèques, cela peut devenir très compliqué.

Pour éviter cette contrainte, le C++ permet de regrouper les noms dans un espace dédié : les espaces de noms (*namespace*). En créant un espace de noms, vous évitez les conflits entre les noms, ce qui peut simplifier vos codes. La bibliothèque standard utilise un espace de noms appelé `std`. Vous apprendrez par la suite à créer des espaces de noms, mais pour l'instant, voyons comment utiliser l'espace de noms de la bibliothèque standard.

Pour utiliser l'objet `cout` de la bibliothèque standard, il faut donc préciser que celui-ci provient de l'espace de noms `std`. Plusieurs solutions sont possibles, selon le contexte. Premièrement, vous pouvez déclarer l'espace de noms `std` à chaque utilisation d'une fonctionnalité de la bibliothèque standard, en utilisant l'opérateur `::` :

main.cpp

```cpp
#include <iostream>

int main() {
    std::cout << "Hello, world!" << std::endl;
}
```

Dans ce cas, il faut faire précéder chaque utilisation de `cout` et de `endl` avec l'espace de noms.

La deuxième solution est de déclarer que vous allez utiliser une fonctionnalité d'un espace de noms en utilisant le mot-clé `using`. Lorsque le compilateur rencontre ensuite `cout`, il saura qu'il faut utiliser l'objet `cout` :

main.cpp
```
#include <iostream>
using std::cout;

int main() {
    cout << "Hello, world!" << std::endl;
}
```

Vous pouvez remarquer ici que seule l'objet `cout` est déclaré en utilisant `using`. L'objet `endl` n'étant pas déclaré de cette manière, il faut l'écrire en utilisant la première syntaxe.

Pour terminer, il est possible d'activer un espace de noms globalement, en utilisant `using namespace`.

main.cpp
```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello, world!" << endl;
}
```

Vous remarquez ici qu'il n'est plus nécessaire d'écrire `std::` devant `cout` et `endl`. Cependant, cette syntaxe peut poser des problèmes, puisqu'il pourra y avoir des conflits entre un nom que vous aurez donné et un nom de la bibliothèque standard. Cela posera en particulier des problèmes lorsque vous utiliserez plusieurs fichiers. Vous verrez dans le chapitre correspondant les règles à utiliser, il est donc préférable de limiter l'utilisation de cette syntaxe et préférer la première syntaxe.

cout est un flux

Comme vous l'avez remarqué dans les codes précédents, pour afficher un message avec `cout`, il faut utiliser l'opérateur `«`. Par exemple, pour afficher le texte “hello”, il faut écrire :

```
cout << "Hello";
```

Ce code ne doit pas être compris comme signifiant “afficher hello”, mais comme “envoyer hello à l'objet cout”. L'opérateur `«` permet d'envoyer des valeurs dans un flux de données.

Il est possible d'envoyer plusieurs valeurs en série dans un flux, en les séparant par plusieurs opérateurs `«` :

```
cout << "Hello" << "world";
```

Ce code signifie “envoyer la chaîne hello à l'objet cout, puis envoyer la chaîne world à l'objet cout”. Ce code est équivalent à appeler plusieurs fois l'objet `cout` :

```
cout << "Hello";
cout << "world";
```

Vous pouvez de cette manière afficher n'importe quelle valeur, aussi bien des chaînes que des entiers ou des nombres réels :

```
cout << "une chaîne" << 123 << 123.456;
```

Exercices

Modifier le code suivant pour afficher les messages demandés. Faire l'exercice

main.cpp

```
#include <iostream>

int main() {
    // Modifier la ligne suivante pour afficher "Bienvenue
tout le monde !"
    std::cout << "Hello, world!" << std::endl;
```

```
    // Ajouter UNE ligne pour afficher "Bienvenue !" et
"Tout le monde !" sur DEUX lignes
    std::cout << @@@@

    // Ajouter DEUX lignes pour afficher "Bienvenue !" et
"Tout le monde !" sur UNE ligne
    std::cout << @@@@
    std::cout << @@@@

    return 0;
}
```

Exos : include, recherche quel include pour utiliser vector ? array ? etc

Aller plus loin

* Le programme Hello World dans différents langage : Wikipédia.

Chapitre précédent	Sommaire principal	Chapitre suivant

Cours, C++